TASK

Django V

Visit our website

Introduction

WELCOME TO THE INTRODUCTION TO DJANGO V TASK!

In this task, we will continue building our poll application. We will cover some important topics such as working with regular expressions, basic error handling and creating forms and templates.

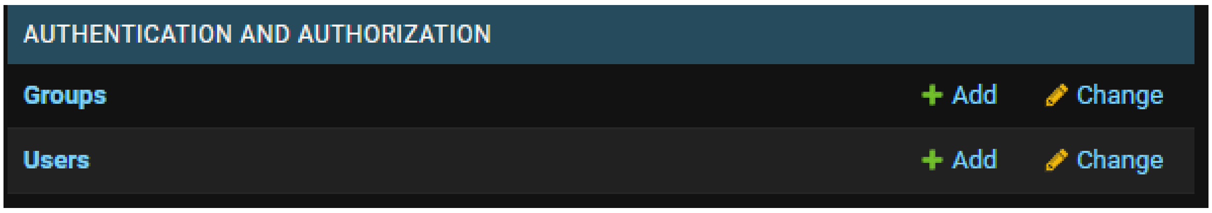
Disclaimer: We've curated the most relevant bits of the official documentation for you and added some additional explanation to make Django as concise and accessible as possible.

ADDING USERS TO OUR POLL APP

Most websites these days are not useful without the option to add users to their databases. This would typically require setting up a user class. In order to remember the user each time round, keeping this state is difficult in HTTP (which is a stateless protocol). Luckily, Django makes this much easier by using a built-in module.

Let's start off relatively easily: just by adding a user to the Users database. You may be thinking, "But I haven't created a Users database"! Django actually ships projects with a Users database by default - isn't this convenient?

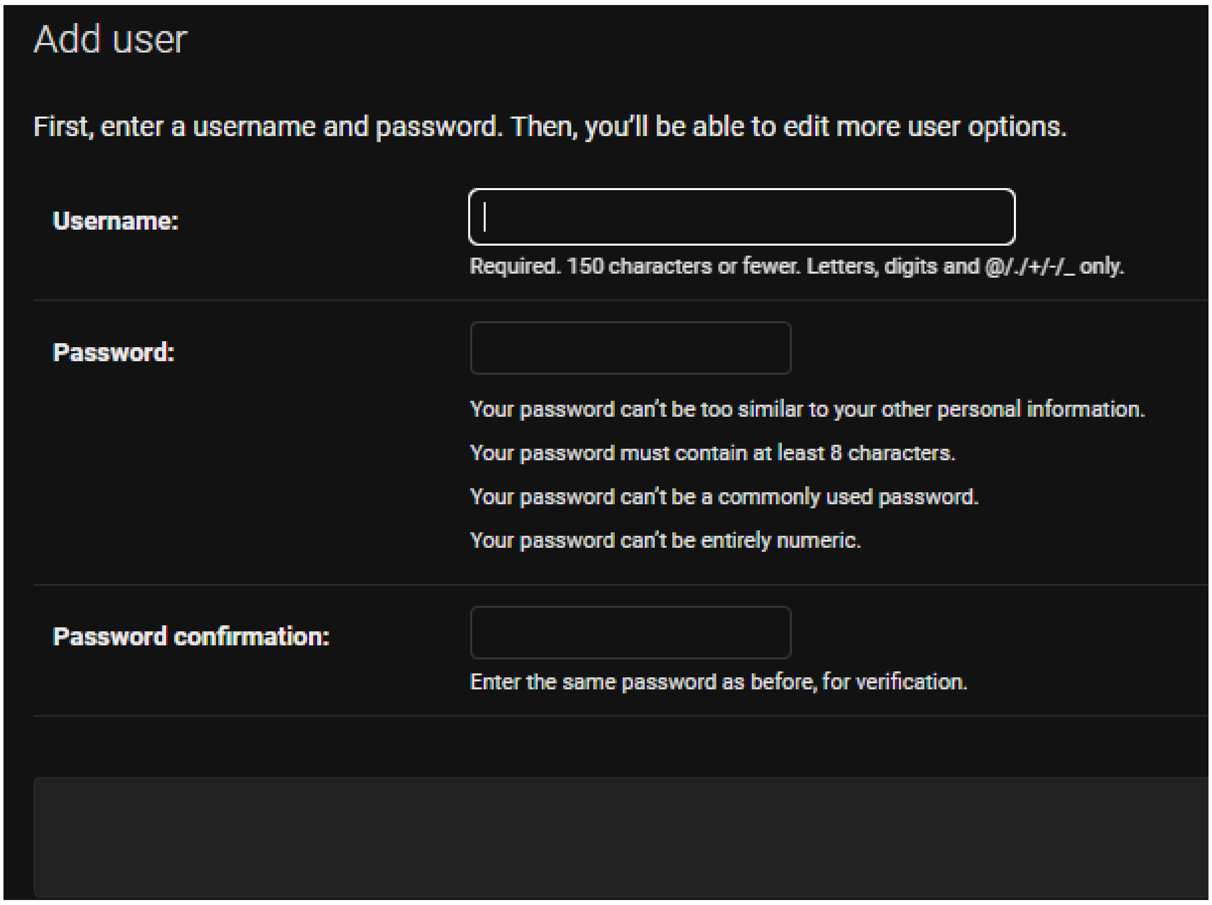
Let's start by starting up the server and navigating to the **admin page**. You should be able to see a set of tables for authentication and authorisation:



AUTHENTICATION AND AUTHORIZATION		
Groups	+ Add	Change
Users	+ Add	Change

The Users table simply is a list of all users. For now, the Groups table can be ignored. This is just something to apply similar permissions for different types of user.

So far, you have created a superuser via the command line (hence you are able to access this page). Now, we can just create a regular, non-administrative user. Just click "Add" for the Users table.


Add user

First, enter a username and password. Then, you'll be able to edit more user options.

Username:

Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.

Password:

Your password can't be too similar to your other personal information.

Your password must contain at least 8 characters.

Your password can't be a commonly used password.

Your password can't be entirely numeric.

Password confirmation:

Enter the same password as before, for verification.

By now, you've probably done this quite a few times. Just enter a username and pick a password: congratulations, this is now your first regular user of the system.

LOGGING IN

Let's add a few more views to polls/views.py. These views are slightly different because they take an additional argument (question_id):

Now that we have our first regular user in the database, let's implement something to log them in. This is simply done as a Django app. Start by setting up an app called user_auth. Set up a path to it in the **hyperion/** directory. But, before doing so, there is a specific line to add in before:

```
path('user_auth/', include("django.contrib.auth.urls")),
path('user_auth/', include("user_auth.urls")),
```

This extra line allows you to access the in-built authentication for Django.

Create a **urls.py** with the following in it:

```
app_name = 'user_auth'
urlpatterns = [
    path('', views.user_login, name='login')
]
```

Now, we want to create a login page for the user. Start by creating a **templates/authentication/login.html** in the **user_auth/** folder. Now, set up your `user_login` view in **views.py** to take the user to the login page:

```
def user_login(request):
    return render(request, 'authentication/login.html')
```

This is simply a form, just like we did in the last task. This will look very much like regular HTML:

```
<h1> Login as User </h1>

<form action="{% url 'user_auth:authenticate_user' %}" method="post">
    {% csrf_token %}

    <label>Username</label>
    <input type="text" name="username" placeholder="User" required>
    <br/><br/>

    <label>Password</label>
    <input type="password" name="password" required>
    <br/><br/>

    <input type="submit" value="Login" />
</form>
```

The more eagle-eyed will notice one small detail: this form submits to a path called `authenticate_user` in the `user_auth` app. We haven't set this up yet, so let's do this. By now, this should be second-nature. Your **user_auth/urls.py** should look like this:

```
app_name = 'user_auth'
urlpatterns = [
    path('', views.user_login, name='login'),
    path('authenticate_user/', views.authenticate_user,
name='authenticate_user')
]
```

Now, what we want is to do one of two things:

- If login is successful, take them to a page to show their details
- If unsuccessful, return to the login page

So, setting up our `show_user` method, let's first see how we can authenticate the user:

```
from django.contrib.auth import authenticate, login


def authenticate_user(request):
    username = request.POST['username']
    password = request.POST['password']
    user = authenticate(username=username, password=password)
```

The `authenticate` method simply looks up in the Users table and returns an object that represents the logged-in user. If the user doesn't exist in the table, this simply returns `None`. Therefore, let's send the user back to login if the object is `None`, and to a new HTML page otherwise:

```
    if user is None:
        return HttpResponseRedirect(
            reverse('user_auth:login')
        )
    else:
        login(request, user)
        return HttpResponseRedirect(
            reverse('user_auth:show_user')
        )
```

So the final method should look like this:

```
def authenticate_user(request):
    username = request.POST['username']
    password = request.POST['password']
    user = authenticate(username=username, password=password)
    if user is None:
        return HttpResponseRedirect(
            reverse('user_auth:login')
        )
    else:
        login(request, user)
        return HttpResponseRedirect(
            reverse('user_auth:show_user')
        )
```

And now this creates one more view: the `show_user` view. This just reads in the user data and sends it to (and renders) a new HTML file. In order to read the user data, this can simply be found in the `request.user` object:

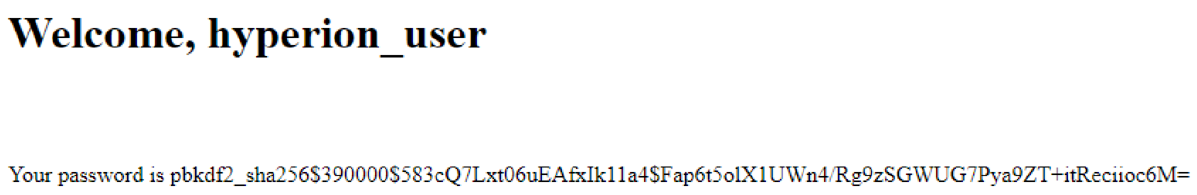
```
def show_user(request):
    print(request.user.username)
    return render(request, 'authentication/user.html', {
        "username": request.user.username,
        "password": request.user.password
    })
```

There is one final piece missing: the **authentication/user.html**. Go ahead and create this file. For now, we will stick to something simple, just to show the username and password of the user:

```
<h1> Welcome, {{username}} </h1>

<p> Your password is {{password}}</p>
```

You will now see something like this:



```
Welcome, hyperion_user

Your password is pbkdf2_sha256$390000$583cQ7Lxt06uEAfxIk11a4$Fap6t5olX1UWn4/Rg9zSGWUG7Pya9ZT+itReciioc6M=
```

You will notice something strange: the password isn’t the one you typed in. This is a great feature of Django. It automatically hashes your passwords for you. This way, you never have to store the user's passwords in plain-text (which is considered dangerous).

INSTRUCTIONS

Feel free at any point to refer back to the material if you get stuck. Remember that if you require more assistance, we are always here to help you!

Compulsory Task

- Make sure you've set up your poll app correctly.
- Create a new **user registration** page. This will require a field for a username, password, and first name.
- Create a login page.
- Extend the poll application so that you can only vote if you have been logged in.

Rate us

Share your thoughts

HyperionDev strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think that the content of this task, or this course as a whole, can be improved, or think we've done a good job?

Click here to share your thoughts anonymously.

REFERENCE

Django documentation. (n.d.). Django Software Foundation. Retrieved October 18, 2022, from **https://docs.djangoproject.com/en/4.1/**